Phong Equation Using Streaming SIMD Extensions

Version 2.1

01/99

Order Number: 243653-002

Table of Contents

1 Introduction 1
2 Phong Reflection Equation 1
2.1 Implementing the Phong Equation Algorithm 3
2.1.1 Data Format 3
2.1.2 Implementation of the SoA Data Layout 4
2.1.3 Implementation of the AoS Data Layout 5
2.2 Optimization Techniques 5
2.2.1 Separating an Implicit Load from an Operation 6
2.2.2 Using the Prefetch Instruction 6
3 Performance 6
3.1 Gains/Improvements 7
4 Conclusion 7
5 C Coding Example 7
6 Streaming SIMD Extensions AoS Assembly Code Example 11
6.1 AoS Unoptimized Version 11
6.2 AoS Optimized Version 19
7 Streaming SIMD Extensions SoA Assembly Code Example 26
7.1 SoA Unoptimized Version 27
7.2 SoA Optimized Version 35

Revision History

Revision	Revision History	Date
2.1	FCS revision.	01/99

References

The following documents are referenced in this application note, and provide background or supporting information for understanding the topics presented in this document.

1. *3D Computer Graphics*, Alan Watt, second edition, pages 89 – 102, (Copyright 1993, Addison-Wesley Publishing Company, Inc.)

2. *Computer Graphics*, Foley, Van Dam, Feiner, Hughes, Second edition in C, pages729 – 731, (Copyright 1996, Addison-Wesley Publishing Company, Inc.)

1 Introduction

Streaming SIMD Extensions for the Intel® Architecture (IA) instruction set provide floating point single-instruction, multiple-data (SIMD) instructions. These single-precision floating point instructions provide a means to accelerate operations typical of 3D graphics, real-time physics, and spatial (3D) audio. This application note illustrates the implementation of Phong Equation using MMX™ Technology and Streaming SIMD Extensions to achieve better performance. This application note also includes examples of code that demonstrate the use of Streaming SIMD Extensions.

2 Phong Reflection Equation

Phong equation is also known as Phong reflection equation. Phong equation is one of the widely used computer graphics reflection models that describes the interaction of light with a surface in terms of the properties of the surface and the nature of the incident light. The purpose of a reflection model in computer graphics is to render three-dimensional objects in two-dimensional space. The general rendering process first represents objects as meshes of planar polygons. The Phong reflection equation is then used to determine the intensities of the vertices of the polygon mesh. The polygons are transformed to two-dimensional space and rendered with certain scanline algorithms. [1][2]

Suppose I is the light intensity of a point on a surface. By the Phong equation, I is calculated as a linear combination of three light components: the ambient light, the diffuse light, and the specular light: This is described by Formula I.

$$I = I_a + I_d + I_s \qquad \text{(I)}$$

The ambient component of the light is defined in Formula II.

$$I_a = I_{am} * K_a \qquad \text{(II)}$$

In Formula II, I_{am} is the incident ambient light intensity, and K_a is the ambient coefficient that represents the ambient property of the surface.

The diffuse component of the light is defined as

$$I_d = I_i * K_d * \cos(\theta)$$

In this definition of the diffuse component, I_i is the incident intensity from a point light source, and K_d is the diffuse reflection constant. The angle θ is the angle between the light direction L(lx,ly,lz) and the surface normal N(Nx,Ny,Nz) as shown in Figure 1. Both L and N are unit vectors, hence $\cos(\theta)$ = dot(L,N). In terms of vector notation, the diffuse light component can be evaluated with Formula III.

$$I_d = I_i * K_d * \text{dot}(L,N) \qquad \text{(III)}$$

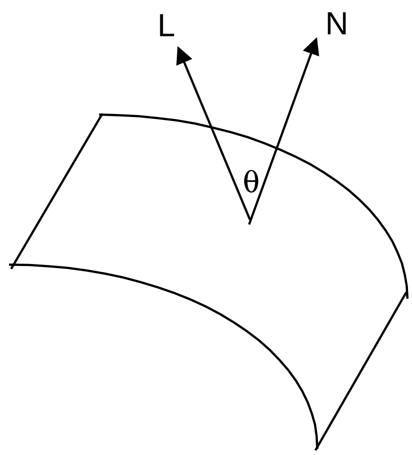

Figure 1: The Diffuse component of the light

The specular component of the light is defined as

$$I_s = I_i * K_s * \cos^n(\varphi)$$

In the definition of the specular component, K_s is a specular reflection constant, and the angle φ is the angle between the viewing direction and the mirror vector R. The term *n* is an index that simulates surface roughness. In term of vector notation, the specular component can be evaluated with Formula IV.

$$I_s = I_i * K_s * (\mathrm{dot}(R,V))^n \tag{IV}$$

In Formula IV, R = 2.0*dot(L,N)N - L

Combining formulas (II), (III) and (IV), the Phong equation then becomes:

$$I = I_{am} * K_a + I_i * (K_d * \mathrm{dot}(L,N) + K_s * (\mathrm{dot}(R,V))^n\ ) \tag{V}$$

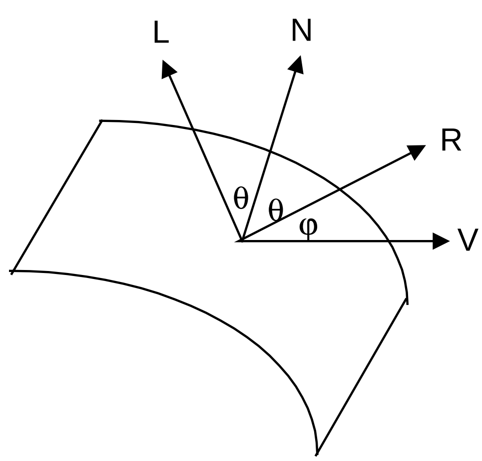

Figure 2: The Specular component of the light

For colored objects, three equations are used to calculate the red, green and blue intensities respectively:

$$Ir = I_{am}*K_{ar} + I_i*K_{dr}*\mathrm{dot}(L,N) + I_iK_s*(\mathrm{dot}(R,V))^n \quad \text{(VI)}$$

$$Ig = I_{am}*K_{ag} + I_i*K_{dg}*\mathrm{dot}(L,N) + I_iK_s*(\mathrm{dot}(R,V))^n \quad \text{(VII)}$$

$$Ib = I_{am}*K_{ab} + I_i*K_{db}*\mathrm{dot}(L,N) + I_iK_s*(\mathrm{dot}(R,V))^n \quad \text{(VIII)}$$

$$R = 2.0*\mathrm{dot}(L,N)N - L \quad \text{(IX)}$$

Formulas (VI) to (IX) are used in the implementation of Phong Equation discussed in the paper.

There are many other ways to implement the Phong Equation. Some of these use approximation to increase performance at the expense of reducing the visual quality (see references[1] and [2]).

2.1 Implementing the Phong Equation Algorithm

This section discusses the methodology for implementing the Phong equation algorithm using MMX technology code and Streaming SIMD Extensions.

2.1.1 Data Format

Three-dimensional vectors are the primary operands used in the Phong equation. The input data for the example code includes vectors with floating point components (x,y,z) that describe the light direction, viewing direction, and surface normals. The input data also includes vectors with floating point components (R,G,B) that describe the ambient and diffuse nature of the light. The outputs are a stream of the color intensities of the vertices. The color intensity is also a vector with byte components (red, green, blue). The following code fragment shows the data structures of the vectors used in the implementation:

```
Struct point_3d{
   float x,y,z;
};

Struct RGB_float{
   float R,G,B;
};

Struct RGB_Color{
   unsigned char R,G,B;
};
```

The input data representing the light, viewing direction, and the ambient and diffuse light components are single vector (constant). The surface normal vectors are a stream of input data. There are two primary data layouts to organize the surface normal vectors to take advantage of a SIMD architecture. These data layouts are referred to as Array of Structures (AoS) and Structure of Arrays (SoA). The following code fragment shows the difference in these two data layouts:

```
Array of Structures:
====================
  struct point3d{
    float x;
    float y;
    float z;
  };
  struct  point3d  normVectors[SIZE];
```

```
Structure of Arrays:
====================
  struct norm_list{
     float  nx[SIZE];
     float  ny[SIZE];
     float  nz[SIZE];
  } Normlist;
```

There are many factors to consider before selecting a data layout. With regard to performance and in general, SIMD operations are most effective on programs that use the SoA data format. The SoA data format typically allows the full width of the SIMD register to be used simultaneously, which maximizes the potential performance gain. Sections 2.1.2 and 2.1.3 discuss the AoS and SoA data layouts in detail. Source code and performance results are provided for both the AoS and SoA implementations in this paper.

2.1.2 Implementation of the SoA Data Layout

With the SoA data layout, the same data elements of the normal vectors are stored together in memory. This type of data layout is well-suited for SIMD operations. Three-dimensional vector operations, such as vector scaling, vector addition/subtraction, and vector dot product, are the major operations in the Phong Equation. With the SoA data layout, vector operations benefit from the Streaming SIMD Extensions because separate components of four vectors can be computed simultaneously.

The Streaming SIMD Extensions implementation of the SoA data layout primarily consists of a data setup section and two major loops (refer to Section 7). In the data setup section, vectors representing light, viewing direction, ambient light component, and diffuse light component are transformed to a planar form to match the planar data of the normal vectors (as depicted in Figure 3). The planar data is stored in a temporary buffer for use later in the loop.

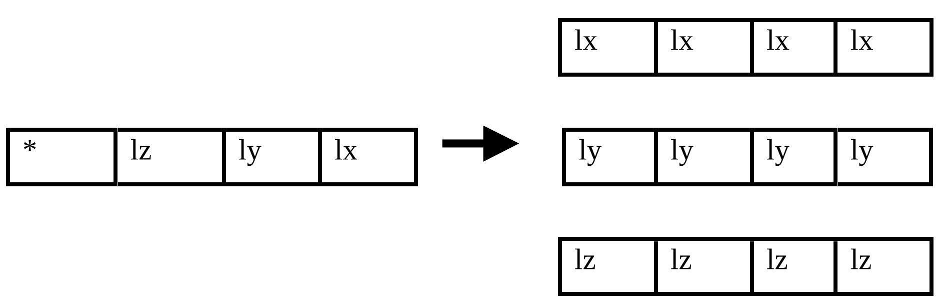

Figure 3: Conversion of the light direction data to planar form

In each iteration of the first loop (labeled as `NextFourVertex`), the planar data of four normal vectors is loaded into three Pentium® III registers as shown in Figure 4. The four components of each register are operated on in parallel during the subsequent vector operations.

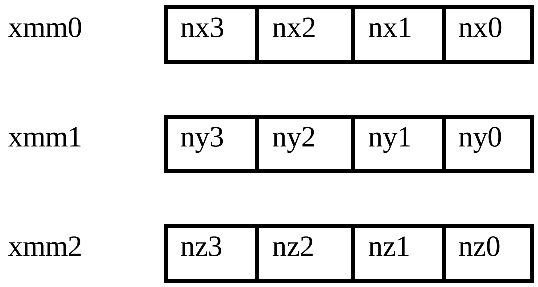

Figure 4: Planar data of four normal vectors

The second loop (labeled as `NextVertex`) is designed for the case where the number of vertices is not a multiple of four. In each iteration of this loop, the data of only one normal vector is loaded. Note that this second loop could be eliminated if the programmer can guarantee that the number of vertices is always a multiple of four; however, this is recognized as a rare case.

2.1.3 Implementation of the AoS Data Layout

The AoS format stores x, y and z coordinates together in memory for a normal vector. Vector operations with three components would not get full benefit from the Pentium® III registers, because one of the four components of the SIMD register is not used. To take the advantage of the full width of the SIMD register as discussed in Section 2.1.2, the AoS-formatted data elements are transformed to a planar format. In the Streaming SIMD Extensions for the AoS data layout (refer to Section 6), each loop iteration loads the data of four normal vectors into Pentium® III registers. The data is then transformed to a planar form with five ***shufps*** instructions as shown in Figure 5. After the transformation, the rest of the Streaming SIMD Extensions for the AoS data layout is almost the same as for the SoA data layout.

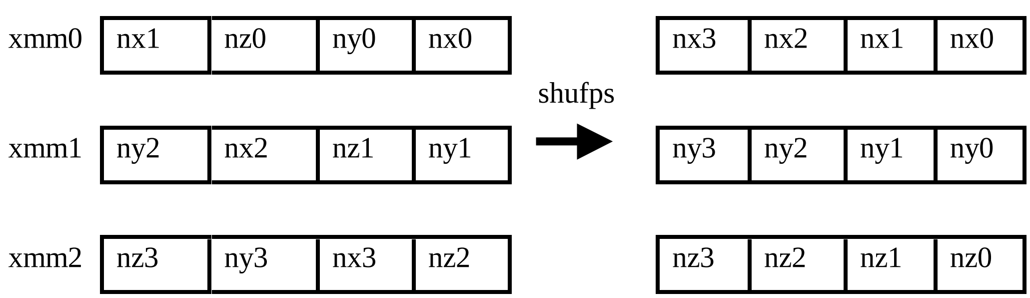

Figure 5: Conversion of four normal vectors data to planar form

2.2 Optimization Techniques

Two techniques were used to optimize the MMX Technology and Streaming SIMD Extensions. These techniques are discussed in the following sections. The code for these optimized implementations of the AoS and SoA data layouts are provided in Sections 6.2 and 7.2.

2.2.1 Separating an Implicit Load from an Operation

Separating an implicit load from a memory-to-register instruction, and moving the load instruction upstream, allows the load instruction to retire before the instruction for the operation. Example 1 shows a multiplication operation from the non-optimized code and optimized code. In the optimized code, one more instruction, ***movaps,*** is used to separate the load from the operation, but the number of micro-ops does not increase. The advantage of separating the load from the operation is that the load can be moved upstream in the instruction flow and retire more quickly, thereby freeing valuable processor resources.

Note that in the optimized code example, some flow of additional instructions must occur between the `movaps` and the `mulps` instructions or the technique does not work.

Unoptimized code:

```
mulps     xmm0, [eax]
```

Optimized code:

```
movaps    xmm5, [eax]
..............
..............  (flow of additional or other instructions)
..............
mulps       xmm0, xmm5
```

Example 1: Optimization technique of separating a load from an operation

2.2.2 Using the Prefetch Instruction

In the optimized code using Streaming SIMD Extensions, a prefetch instruction is used in the loop to fetch the next cache line of data from the input normal vectors in advance (conventionally preloading means loading a value into a register). The results from simulating this code indicate that by using the prefetch instruction, it is possible to achieve a 36% speedup, with the assumption that the data of the normal vectors is not in cache.

Prefetch is an important feature of the Streaming SIMD Extensions; however, detailed coverage of prefetching techniques is beyond the scope of this paper. For more detailed information about the prefetch instruction and techniques for code optimization, refer to the following documents (contact your Intel representative for availability of the documents):

Intel® Architecture Optimization Manual, Order No: TBD

Intel® Application Note, *Achieving High-Performance 3D Geometry on Pentium® III processor* , AP-816, Order No: 243650-001

3 Performance

The primary comparison this paper offers is between an AoS and a SoA data layout implementation using Streaming SIMD Extensions assembly code.

3.1 Gains/Improvements

Performance improvement achieved by the Streaming SIMD Extensions results from a combination of SIMD operations on the data and use of the prefetch instruction. Furthermore, the data layout is shown to be an important consideration in overall code performance: SoA proves better than AoS because an additional seven instructions are needed to shuffle the AoS data of four vectors to the SoA format in each loop iteration. For more information on the AoS and SoA data formats, refer to the *RGB and RGBA Data Swizzling Using MMX™ Technology*, Intel application note (AP-813, Order No.: 243647-001).

4 Conclusion

The Phong Equation is a well-known algorithm for producing a realistic shading effect in 3D computer graphics, but it is computationally expensive. The efficiency of this algorithm is critical to the performance of a 3D graphics/geometry applications because the algorithm typically must process several thousand vertices (or more) in the polygon meshes that comprise even simple 3D scenes. Intel's MMX™ Technology cannot be applied to the Phong equation algorithm because of its floating-point nature. The Streaming SIMD Extensions provide floating point SIMD operations that can be used to achieve better performance for the Phong equation algorithm. Using a Streaming SIMD Extensions implementation provides significant improvement compared to a C implementation.

5 C Coding Example

This section contains the C implementation of the Phong equation in both the AoS and SoA data layouts.

```
//*************************************************************************
//  This function computes the RGB intensities of a series of surface point
//  using the Phong reflection equation:
//
//     I = Iam*Ka + Ii*(Kd*dot(L,N) + Ks*(dot(R,V))n )
//     R = 2.0*dot(L,N)N - L
//
//  INPUTs:
//    int num_pnt
//     number of surface points of which RGB intensities will to be
calculated.
// Point_3d *light
//     Pointer pointing to a (lx,ly,lz) vector representing light
direction.
// Point_3d *norm
//     Pointer pointing to a list of vectors (nx, ny,nz) representing
normal
//     vectors of the surface points.
//  Point_3d *View
//     Viewing  direction
//F_RGB *ambient
```

```
//   Pointer pointing to a (R,G,B) vector representing the coefficients of
RGB
//   ambient components, where
//   ambient->R = Ia*Kar, ambient->G = Ia*Kag, ambient->B = Ia*Kab
// F_RGB *diffuse
//   Pointer pointing to a (R,G,B) vector representing coefficients of RGB
//   diffuse components, where
//   diffuse->R = Ii*Kdr, diffuse->G = Ii*Kdg, diffuse->B = Ii*Kdb
// float specular
//   Coefficient of specular where
//   specular = Ii*Ks
// long phongconst
//   Phong constant
//
//  OUTPUT
// struct RGB_color *rgb
//   Pointer pointing to a list of (r,g,b) vectors representing the RGB
intensities
//of the points of the surface.
// Vectors used in this function:
// typedef struct point3d
// {
//       float x,y,z;
//   }Point_3d;
//
// typedef struct RGB_float
// {
//   float R,G,B;
// }F_RGB;
//
// struct RGB_color
// {
//   unsigned char R,G,B;
// }
//
//****************************************************************/
void intensitys_AoS(
       int num_pnt,
       Point_3d *norm,
       Point_3d *ldir,
       Point_3d *v_pnt,
       F_RGB *ambient,
       F_RGB *diffuse,
       float specular,
```

```
        long phongconst,
        struct RGB_color *rgb)
{
   int i, j;
   float dt, costheta, cosalpha,specularN;
   float intensityR,intensityG, intensityB;
float lx = ldir->x,ly = ldir->y,lz = ldir->z;
   float vx = v_pnt->x,vy = v_pnt->y,vz = v_pnt->z;
   Point_3d R;

   for(j = 0; j < num_pnt; j++){
     costheta = norm->x*lx + norm->y*ly + norm->z*lz;

     // Calculate the mirror vector: R = 2*(dot(N,L)N - L
     dt = 2.0f * costheta;
     R.x = dt * norm->x - ldir->x;
     R.y = dt * norm->y - ldir->y;
     R.z = dt * norm->z - ldir->z;

     if(costheta < 0.0)   // back face
       costheta = -costheta;
     // Sum up ambient and diffuse components
     intensityR = ambient->R + diffuse->R * costheta;
     intensityG = ambient->G + diffuse->G * costheta;
     intensityB = ambient->B + diffuse->B * costheta;

     // Calculate specular component
     cosalpha = R.x*vx + R.y*vy + R.z*vz;
     if (cosalpha > 0.0){
       specularN = specular;
       for(i = 0; i < phongconst; i++)
         specularN *= cosalpha;
       intensityR += specularN;
       intensityG += specularN;
       intensityB += specularN;
     }

     // Convert the RGB intensities from floats to bytes
rgb->R = min(255,(intensityR * 255.0f + 0.5f));
     rgb->G = min(255,(intensityG * 255.0f + 0.5f));
     rgb->B = min(255,(intensityB * 255.0f + 0.5f));
     norm++;
     rgb++;
   }
```

```
}

//**************************************************************************
  // For Input and output data structure, refer to
// the header of intensitysAoS()
//
  //  normal vectors are organized as SoA data layout as followings:
     //
// struct Vertex
  //{
  //   float  normx[SIZE];
  //   float  normy[SIZE];
  //   float  normz[SIZE];
  //...................;
  //}NormList;
  //
// Points of normx, normy and normz are passed to the
// function
//**************************************************************************

void intensitys_SoA(
       int num_pnt,
       float *normx,
       float *normy,
       float *normz,
       Point_3d *ldir,
       Point_3d *v_pnt,
       F_RGB *ambient,
       F_RGB *diffuse,
       float specular,
       long phongconst,
       struct RGB_color *rgb)
{
  int i, j;
  float dt, costheta, cosalpha,specularN;
  float intensityR,intensityG, intensityB;
float lx = ldir->x,ly = ldir->y,lz = ldir->z;
  float vx = v_pnt->x,vy = v_pnt->y,vz = v_pnt->z;
  Point_3d R;

  for(j = 0; j < num_pnt; j++){
    costheta = normx[j]*lx + normy[j]*ly + normz[j]*lz;
    // Calculate the mirror vector: R = 2*(dot(N,L)N - L
```

```
    dt = 2.0f * costheta;
    R.x = dt * normx[j] - lx;
    R.y = dt * normy[j] - ly;
    R.z = dt * normz[j] - lz;

    if(costheta < 0.0)   // back face
      costheta = -costheta;
    // Sum up ambient and diffuse components
    intensityR = ambient->R + diffuse->R * costheta;
    intensityG = ambient->G + diffuse->G * costheta;
    intensityB = ambient->B + diffuse->B * costheta;

    // Calculate specular component
    cosalpha = R.x*vx + R.y*vy + R.z*vz;
    if (cosalpha > 0.0){
      specularN = specular;
      for(i = 0; i < phongconst; i++)
        specularN *= cosalpha;
      intensityR += specularN;
      intensityG += specularN;
      intensityB += specularN;
    }

    // Convert the RGB intensities from floats to bytes
rgb->R = min(255,(intensityR * 255.0f + 0.5f));
    rgb->G = min(255,(intensityG * 255.0f + 0.5f));
    rgb->B = min(255,(intensityB * 255.0f + 0.5f));
    rgb++;
  }

}
```

6 Streaming SIMD Extensions AoS Assembly Code Example

The following sections provide code samples for both an unoptimized and an optimized version of the Phong equation algorithm using the AoS data layout.

6.1 AoS Unoptimized Version

```
void Intensitys_AOS_XMM(int num_pnt,
            Point_3d *norm,
            Point_3d *light,
            Point_3d *View,
            F_RGB *ambient,
```

```
            F_RGB *diffuse,
            float   specular,
            long phongconst,
            struct RGB_color *rgb)
{
  float buffer[56];
  float *tem;
  int reNum;

  tem = (float *) (((unsigned)buffer + 15) & ~15);
__asm{
  mov       eax,000FF00FFh
  mov       esi, light
  mov       edi, View
  mov       edx, tem
  mov       ecx, num_pnt     // ecx = num_pnt
  movd   mm7, eax         // mm7 = [0,0,255,255]
  punpcklwd mm7, mm7      // mm7 = [255,255,255,255]

  movups    xmm0, [esi]      // xmm0 = [*,lz,ly,lx]
  movaps    xmm1, xmm0    // xmm1 = [*,lz,ly,lx]
  movaps    xmm2, xmm0    // xmm2 = [*,lz,ly,lx]
  mov       eax, diffuse
  mov       esi, ambient

  shufps    xmm0, xmm0, 0    // xm0 = [lx,lx,lx,lx]
  shufps    xmm1, xmm1, 85   // xm1 = [ly,ly,ly,ly]
  shufps    xmm2, xmm2, 170  // xm2 = [lz,lz,lz,lz]

  movaps    [edx], xmm0
  movaps    [edx+16], xmm1
  movaps    [edx+32], xmm2

  movups    xmm3, [edi]      // xmm3 = [*,vz,vy,vx]
  movaps    xmm4, xmm3    // xmm4 = [*,vz,vy,vx]
  movaps    xmm5, xmm3    // xmm5 = [*,vz,vy,vx]

  shufps    xmm3, xmm3, 0    // xmm3 = [vx,vx,vx,vx]
  shufps    xmm4, xmm4, 85   // xmm4 = [vy,vy,vy,vy]
  shufps    xmm5, xmm5, 170  // xmm5 = [vz,vz,vz,vz]

  movaps    [edx+48], xmm3
  movaps    [edx+64], xmm4
  movaps    [edx+80], xmm5
```

```
  movups    xmm0, [esi]      // xmm0 = [*,ambientB,ambientG,ambientR]
  movaps    xmm1, xmm0     // xmm1 = [*,ambientB,ambientG,ambientR]
  movaps    xmm2, xmm0     // xmm2 = [*,ambientB,ambientG,ambientR]

  shufps    xmm0, xmm0, 0    // xmm0 = [ambientR,ambientR,ambientR,ambientR]
  shufps    xmm1, xmm1, 85   // xmm1 = [ambientG,ambientG,ambientG,ambientG]
  shufps    xmm2, xmm2, 170    // xmm2 =
[ambientB,ambientB,ambientB,ambientB]

  movaps    [edx+96], xmm0
  movaps    [edx+112], xmm1
  movaps    [edx+128], xmm2

  movups    xmm3, [eax]      // xmm3 = [*,diffuseB,diffuseG,diffuseR]
  movaps    xmm4, xmm3     // xmm4 = [*,diffuseB,diffuseG,diffuseR]
  movaps    xmm5, xmm3     // xmm5 = [*,diffuseB,diffuseG,diffuseR]

  shufps    xmm3, xmm3, 0    // xmm3 = [diffuseR,diffuseR,diffuseR,diffuseR]
  shufps    xmm4, xmm4, 85   // xmm4 = [diffuseG,diffuseG,diffuseG,diffuseG]
  shufps    xmm5, xmm5, 170    // xmm5 =
[diffuseB,diffuseB,diffuseB,diffuseB]

  movaps    [edx+144],xmm3 // [diffuseR,diffuseR,diffuseR,diffuseR]
  movaps    [edx+160],xmm4 // [diffuseG,diffuseG,diffuseG,diffuseG]
  movaps    [edx+176],xmm5 // [diffuseB,diffuseB,diffuseB,diffuseB]

  movss     xmm6, specular   // xmm6 = [0.0,0.0,0.0,specular]
  shufps    xmm6, xmm6, 0    // xmm6 = [specular,specular,specular,specular]
  movaps    [edx+192], xmm6  // [specular,specular,specular,specular]

  mov       eax, ecx         // eax = num_pnt
  shr       ecx, 2           // ecx = num_pnt/4
  mov       edi, ecx         // edi = num_pnt/4
  shl       edi, 2             // edi = 4*(num_pnt/4)
  sub       eax, edi         // eax = num_pnt - 4*(num_pnt/4)
  mov       edi, rgb
  mov       esi, norm
  mov       reNum, eax       // number of vertices remains
  cmp       ecx, 0
  je        DoRemainder

NextFourVertex:
  movaps xmm2, [esi]         // xmm2 = [nx1,nz0,ny0,nx0]
```

```
  movaps xmm3, [esi+16]     // xmm3 = [ny2,nx2,nz1,ny1]
  movaps xmm4, [esi+32]     // xmm4 = [nz3,ny3,nx3,nz2]

  movaps xmm0, xmm2       // xmm0 = [nx1,nz0,ny0,nx0]

  shufps xmm2, xmm3,73    // xmm2 = [nz1,ny1,nz0,ny0], 0100 1001
  shufps xmm3, xmm4,158   // xmm3 = [ny3,nx3,ny2,nx2], 1001 1110
  movaps xmm1, xmm2       // xmm1 = [nz1,ny1,nz0,ny0]

  shufps xmm2, xmm4,205   // xmm2 = [nz3,nz2,nz1,nz0], 1100 1101
  shufps xmm1, xmm3, 216    // xmm1 = [ny3,ny2,ny1,ny0], 1101 1000
  shufps xmm0, xmm3, 140    // xmm0 = [nx3,nx2,nx1,nx0], 1000 1100

  movaps    xmm5, xmm2    // xmm5 = [nz3,nz2,nz1,nz0]
  movaps    xmm4, xmm1    // xmm4 = [ny3,ny2,ny1,ny0]
  movaps    xmm3, xmm0    // xmm3 = [nx3,nx2,nx1,nx0]

  // costheta = dot(norm,ldir)
  mulps     xmm0, [edx]      / / xmm0 = [nx3*lx,nx2*lx,nx1*lx,nx0*lx]
  mulps     xmm1, [edx+16]   // xmm1 = [ny3*ly,ny2*ly,ny1*ly,ny0*ly]
  mulps     xmm2, [edx+32]   // xmm2 = [nz3*lz,nz2*lz,nz1*lz,nz0*lz]
  addps     xmm0, xmm1       // xmm0 = [nx3*lx+ny3*ly, ..,..,..]
  addps     xmm0, xmm2       // xmm0 =
[costheta3,costheta2,costheta1,costheta0]

  movaps    xmm6, xmm0    // xmm6 =
[costheta3,costheta2,costheta1,costheta0]

  // calculate the reflection vector R
  addps     xmm0, xmm0       // xmm0 = [dt3,dt2,dt1,dt0], dti =
2.0*costhetai
  mulps     xmm3, xmm0       // xmm3 = [dt3*nx3,dt2*nx2,dt1*nx1,dt0*nx0]
  mulps     xmm4, xmm0       // xmm4 = [dt3*ny3,dt2*ny2,dt1*ny1,dt0*ny0]
  mulps     xmm5, xmm0       // xmm5 = [dt3*nz3,dt2*nz2,dt1*nz1,dt0*nz0]
  subps     xmm3, [edx]         // xmm3 = [Rx3,Rx2,Rx1,Rx0]

  subps     xmm4, [edx+16]   // xmm4 = [Ry3,Ry2,Ry1,Ry0]

  subps     xmm5, [edx+32]   // xmm5 = [Rz3,Rz2,Rz1,Rz0]

  // cosalpha = dot(R,v_pnt)
  mulps     xmm3, [edx+48]   // xmm3 = [Rx3*Vx,Rx2*vx,Rx1*vx,Rx0*vx]
  mulps     xmm4, [edx+64]   // xmm4 = [Ry3*Vy,Ry2*vy,Ry1*vy,Ry0*vy]
  mulps     xmm5, [edx+80]   // xmm5 = [Rz3*Vz,Rz2*vz,Rz1*vz,Rz0*vz]
  addps     xmm3, xmm4       // xmm3 = [Rx3*Vx+Ry3*Vy,..,..,..]
```

```
  addps     xmm5, xmm3       // xmm5 = [cosalpha3, cosalpha2, cosalpha1,
cosalpha0]

  // set Costheta = |costheta|
  xorps     xmm7, xmm7       // clear xmm7
  subps     xmm7, xmm6       // xmm6 = [-costheta3,-costheta2,-costheta1,-
costheta0]
  maxps     xmm7, xmm6     // xmm7 =
[Costheta3,Costheta2,Costheta1,Costheta0]

  movaps    xmm0,[edx+144] // xmm0 = [diffuseR,diffuseR,diffuseR,diffuseR]
  movaps    xmm1,[edx+160] // xmm1 = [diffuseG,diffuseG,diffuseG,diffuseG]
  movaps    xmm2,[edx+176] // xmm2 = [diffuseB,diffuseB,diffuseB,diffuseB]
  movaps    xmm3,[edx+96]    // xmm3 = [ambientR,ambientR,ambientR,ambientR]
  movaps    xmm4,[edx+112] // xmm4 = [ambientG,ambientG,ambientG,ambientG]
  movaps    xmm6,[edx+128] // xmm6 = [ambientB,ambientB,ambientB,ambientB]

  mulps     xmm0, xmm7       // xmm0 = [diffuseR*Costheta3,..,..,..]
  mulps     xmm1, xmm7       // xmm1 = [diffuseG*Costheta3,..,..,..]
  mulps     xmm2, xmm7       // xmm2 = [diffuseB*Costheta3,..,..,..]
  addps     xmm0,xmm3      // xmm0 = [diffuseR*Costheta3+ambientR,..,..,..]
  addps     xmm1,xmm4      // xmm1 = [diffuseG*Costheta3+ambientG,..,..,..]
  addps     xmm2,xmm6      // xmm2 = [diffuseB*Costheta3+ambientB,..,..,..]

  xorps     xmm6, xmm6       // xmm6 = [0.0, 0.0, 0.0, 0.0]
  movaps xmm7,[edx+192]    // xmm7 = [specular,specular,specular,specular]

  cmpltps   xmm6,xmm5    // 0.0 < cosalpha
  movmskps  eax, xmm6
  cmp       eax, 0
  je      Continue

  andps     xmm5,xmm6      // xmm5 = [cosalphai3,..,..,..], cosai = (0.0 <
cosalphai)? cosalphai:0.0
  mov       eax, phongconst

loop11:
  mulps     xmm7, xmm5       // xmm7 = [specular*cosalpha3,..,..,..]
  dec       eax
  jnz       loop11

  addps     xmm0, xmm7       // xmm0 = [fr3,fr2,fr1,fr0]

  addps     xmm1, xmm7       // xmm1 = [fg3,fg2,fg1,fg0]
```

```
   addps     xmm2, xmm7       // xmm2 = [fb3,fb2,fb1,fb0]

Continue:
   mulps     xmm0, four_255   // xmm0 = [fR3,fR2,fR1,fR0]
   mulps     xmm1, four_255   // xmm1 = [fG3,fG2,fG1,fG0]
   mulps     xmm2, four_255   // xmm2 = [fB3,fB2,fB1,fB0]

   // Interleave R,G,B
   movaps    xmm3, xmm0       // xmm3 = [fR3,fR2,fR1,fR0]
   shufps    xmm0, xmm1,136   // xmm0 = [fG2,fG0,fR2,fR0]
   shufps    xmm3, xmm2,141   // xmm3 = [fB2,fB0,fR3,fR1]
   shufps    xmm1, xmm2,221   // xmm1 = [fB3,fB1,fG3,fG1]

   movaps    xmm4, xmm1     // xmm4 = [fB3,fB1,fG3,fG1]
   shufps    xmm1, xmm0,216 // xmm1 = [fG2,fR2,fB1,fG1]
   shufps    xmm0, xmm3,40    // xmm0 = [fR1,fB0,fG0,fR0]
   shufps    xmm3, xmm4,215   // xmm3 = [fB3,fG3,fR3,fB2]

   cvtps2pi  mm0, xmm0      // mm0 = [G0,R0]
   cvtps2pi  mm1, xmm1      // mm1 = [B1,G1]
   cvtps2pi  mm2, xmm3      // mm2 = [R3,B2]

   shufps    xmm0,xmm0,14     // xmm0 = [*,*,fR1,fB0]
   shufps    xmm1,xmm1,14     // xmm1 = [*,*,fG2,fR2]
   shufps    xmm3,xmm3,14     // xmm3 = [*,*,fB3,fG3]

   cvtps2pi  mm3, xmm0      // mm3 = [R1,B0]
   cvtps2pi  mm4, xmm1      // mm4 = [G2,R2]
   cvtps2pi  mm5, xmm3      // mm5 = [B3,G3]

   packssdw  mm0,mm3      // mm0 = [R1,B0,G0,R0]
   packssdw  mm1,mm4      // mm4 = [G2,R2,B1,G1]
   packssdw  mm2,mm5      // mm2 = [B3,G3,R3,B2]

   pminsw    mm0, mm7    // mm0 =
[min(255,R1),min(255,B0),min(255,G0),min(255,R0)]
   pminsw    mm1, mm7    // mm1 =
[min(255,G2),min(255,R2),min(255,B1),min(255,G1)]
   pminsw    mm2, mm7    // mm2 =
[min(255,B3),min(255,G3),min(255,R3),min(255,B3)]

   packuswb  mm0, mm1       // mm0 = [G2,R2,B1,G1,R1,B0,G0,R0]

   packuswb  mm2, mm2       // mm2 = [*,*,*,*,B3,G3,R3,B2]
```

```
  movq   [edi], mm0
  movd   [edi+8], mm2

  add      esi, 48       // Next four normal
  add      edi, 12

  dec    ecx
  jne    NextFourVertex

DoRemainder:
  mov    ecx, reNum
  cmp    ecx, 0
  je   fininsh

  // Data set up for the rest of vertices.
  push edx
  mov    eax, 0FFFFFFFh
  movd mm2, eax

  mov    eax, View
  mov    edx, light
  movups xmm0, [eax]
  movups xmm4, [edx]      // xmm4 = [*,lz,ly,lx]

  mov    eax, diffuse
  mov    edx, ambient
  movups xmm3, [eax]      //  xmm3 = [*,diffuse->B,diffuse->G,diffuse->R]
  movups xmm2, [edx]      // xmm2 = [*,ambient->B,ambient->G,ambient->R]

  pop    edx
  movaps [edx], xmm0      // save view in a memory buffer

NextVertex:
  // cosTheta = dot(N,L) = lx*nx+ly*ny + lz*nz
  movups xmm0, [esi]      // xmm0 = [*,nz,ny,nx]
  movaps xmm1, xmm0       // xmm1 = [*,nz,ny,nx]
  mulps  xmm0, xmm4         // xmm0 = [*, lz*nz, ly*ny, lx*nx]

  movaps xmm6, xmm0       // xmm6 = [*, lz*nz, ly*ny, lx*nx]
  shufps xmm0, xmm0,9       //  xmm0 = [*, lx*nx, lz*nz, ly*ny]
  addps  xmm0, xmm6         // xmm0 = [*,
lx*nx+lz*nz,lz*nz+ly*ny,lx*nx+ly*ny]
```

```
  shufps xmm6, xmm6,18       //  xmm6 = [*, ly*ny, lx*nx, lz*nz]
  addps  xmm0, xmm6          // xmm0 = [*, cosTheta, cosTheta, cosTheta]
  movaps xmm7, xmm0        // xmm7 = [*, cosTheta, cosTheta, cosTheta]

  // Calculate the mirror vector: R = 2*(dot(N,L)N - L
  addps  xmm0, xmm0          //  xmm0 = [*,dt,dt,dt], dt = 2*cosTheta
  mulps  xmm0, xmm1          //  xmm0 = [*,dt*nz,dt*ny,dt*nx]
  subps  xmm0, xmm4          //  xmm0 = [*,dt*nz-lz,dt*ny-ly,dt*nx-lx]

  // calculate cosAlpha = cos(Alpha), where
  // Alpha is the angle between view vector V and mirror vector R
  mulps  xmm0, [edx]         //  xmm0 = [*, Rz*Vz,Ry*Vy,Rx*Vx]

  movaps xmm6, xmm0        //  xmm6 = [*, Rz*Vz,Ry*Vy,Rx*Vx]
  shufps xmm0, xmm0,9        //  xmm0 = [*, Rx*Vx, Rz*Vz,Ry*Vy]
  addps  xmm0, xmm6          //  xmm0 = [*, Rx*Vx + Rz*Vz,
Rz*Vz+Ry*Vy,Ry*Vy+Rx*Vx]
  shufps xmm6, xmm6,18       //  xmm6 = [*, Ry*Vy,Rx*Vx,Rz*Vz]
  addps  xmm0, xmm6          // xmm0 = [*, cosAlpha,cosAlpha,cosAlpha]

  // Add Diffuse components
  // If cosTheta < 0.0, the point is on back side of the surface,
  xorps  xmm5, xmm5          // xmm5 = [0.0, 0.0, 0.0, 0.0]
  comiss xmm7, xmm5          // cosTheta < 0.0?  (If True CF = 1)
  jnc    Continue2
  subps  xmm5, xmm7          // xmm5 = [-cosTheta,-cosTheta,-cosTheta,-
cosTheta]
  movaps xmm7,xmm5      // xmm7 = [-cosTheta,-cosTheta,-cosTheta,-cosTheta]

Continue2:
  mulps  xmm7, xmm3          // xmm7 = [*,cosTheta*diffuseB,..,..]
  //  Add Ambient components
  addps  xmm7, xmm2          // xmm7 = Ambient + Diffuse

  // Add Specular components
  xorps    xmm6, xmm6        // clear xmm6
  comiss xmm0, xmm6        // cosAlpha > 0.0 ? (If cosAlpha > 0.0, CF = 0)
  jc   Convert2            // If 0.0 > cosAlpha, no Specular components.
  mov    edx, phongconst
  movss  xmm6, specular      // xmm6 = [0.0,0.0,0.0,specular]

loop21:
  mulss  xmm6, xmm0          // xmm6 = [*,*,*, cosn*cosAlpha]
  dec    edx
```

```
   jnz     loop21

   shufps  xmm6,xmm6,0      // xmm6 = [*,spN,spN,spN]
   addps   xmm7, xmm6          // xmm7 = [*, fB, fG, fR]

Convert2:
   // Transform fR,fG,fB from range 0.0 -- 1.0 to range 0.0 -- 255.0
   mulps   xmm7, four_255     // xmm7 = [*, 255*fB, 255*fG, 255*fR]

   // Convert (fR,fG,fB) from packed floats to packed bytes (R,G,B)
   // and write only three bytes memory location specified by [edi]
   cvtps2pi  mm0, xmm7      // mm0 = [G,R]
   shufps    xmm7, xmm7,2     // xmm1 = [*,*,*,B]
   cvtps2pi  mm1, xmm7      // mm0 = [*,B]
   packssdw  mm0,mm1     // mm0 = [*,B,G,R]
   pminsw  mm0,mm7
   packuswb  mm0,mm0     // mm0 = [*,*,*,*,*,B,G,R]

   // write three bytes output of mm0 to memory [edi]
   maskmovq  mm0, mm2

   add       esi, 12
   add       edi, 3
   dec       ecx
   jne          NextVertex

finish:

   emms
   }


}
```

6.2 AoS Optimized Version

```
void Intensitys_AOS_XMM(int num_pnt,
            Point_3d *norm,
            Point_3d *light,
            Point_3d *View,
            F_RGB *ambient,
            F_RGB *diffuse,
```

```
          float   specular,
          long phongconst,
          struct RGB_color *rgb)
{
  float buffer[56];
  float *tem;
  int reNum;

  tem = (float *) (((unsigned)buffer + 15) & ~15);
__asm{
  mov       eax,000FF00FFh
  mov       esi, light
  mov       edi, View
  mov       edx, tem
  mov       ecx, num_pnt   // ecx = num_pnt
  movd      mm7, eax    // mm7 = [0,0,255,255]
  movups    xmm0, [esi]    // xmm0 = [*,lz,ly,lx]
  mov       eax, diffuse
  mov       esi, ambient
  punpcklwd mm7, mm7       // mm7 = [255,255,255,255]
  movaps    xmm1, xmm0     // xmm1 = [*,lz,ly,lx]
  movaps    xmm2, xmm0     // xmm2 = [*,lz,ly,lx]
  shufps    xmm0, xmm0, 0  // xm0 = [lx,lx,lx,lx]
  movups    xmm3, [edi]      // xmm3 = [*,vz,vy,vx]
  shufps    xmm1, xmm1, 85 // xm1 = [ly,ly,ly,ly]
  shufps    xmm2, xmm2, 170  // xm2 = [lz,lz,lz,lz]
  movaps    [edx], xmm0
  movaps    [edx+16], xmm1
  movaps    [edx+32], xmm2
  movaps    xmm4, xmm3     // xmm4 = [*,vz,vy,vx]
  movaps    xmm5, xmm3     // xmm5 = [*,vz,vy,vx]
  shufps    xmm3, xmm3, 0  // xmm3 = [vx,vx,vx,vx]
  movups    xmm0, [esi]      // xmm0 = [*,ambientB,ambientG,ambientR]
  shufps    xmm4, xmm4, 85 // xmm4 = [vy,vy,vy,vy]
  shufps    xmm5, xmm5, 170  // xmm5 = [vz,vz,vz,vz]
  movaps    [edx+48], xmm3
  movaps    [edx+64], xmm4
  movaps    [edx+80], xmm5
  movaps    xmm1, xmm0     // xmm1 = [*,ambientB,ambientG,ambientR]
  movaps    xmm2, xmm0     // xmm2 = [*,ambientB,ambientG,ambientR]
  shufps    xmm0, xmm0, 0  // xmm0 = [ambientR,ambientR,ambientR,ambientR]
  movups    xmm3, [eax]      // xmm3 = [*,diffuseB,diffuseG,diffuseR]
  shufps    xmm1, xmm1, 85 // xmm1 = [ambientG,ambientG,ambientG,ambientG]
```

```
    shufps    xmm2, xmm2, 170  // xmm2 = [ambientB,ambientB,ambientB,ambientB]

    movaps    [edx+96], xmm0
    movaps    [edx+112], xmm1
    movaps    [edx+128], xmm2
    movaps    xmm4, xmm3     // xmm4 = [*,diffuseB,diffuseG,diffuseR]
    movaps    xmm5, xmm3     // xmm5 = [*,diffuseB,diffuseG,diffuseR]
    shufps    xmm3, xmm3, 0  // xmm3 = [diffuseR,diffuseR,diffuseR,diffuseR]
    movss       xmm6, specular // xmm6 = [0.0,0.0,0.0,specular]
    shufps    xmm4, xmm4, 85 // xmm4 = [diffuseG,diffuseG,diffuseG,diffuseG]
    shufps    xmm5, xmm5, 170  // xmm5 = [diffuseB,diffuseB,diffuseB,diffuseB]
    movaps    [edx+144],xmm3 // [diffuseR,diffuseR,diffuseR,diffuseR]
    movaps    [edx+160],xmm4 // [diffuseG,diffuseG,diffuseG,diffuseG]
    movaps    [edx+176],xmm5 // [diffuseB,diffuseB,diffuseB,diffuseB]
    shufps    xmm6, xmm6, 0  // xmm6 = [specular,specular,specular,specular]
    movaps    [edx+192], xmm6  // [specular,specular,specular,specular]

    mov       eax, ecx         // eax = num_pnt
    shr       ecx, 2           // ecx = num_pnt/4
    mov       edi, ecx         // edi = num_pnt/4
    shl       edi, 2             // edi = 4*(num_pnt/4)
    sub       eax, edi         // eax = num_pnt - 4*(num_pnt/4)
    mov       edi, rgb
    mov       esi, norm
    mov       reNum, eax       // number of vertices remains
    cmp       ecx, 0
    je      DoRemainder

  NextFourVectex:
    movaps xmm2, [esi]         // xmm2 = [nx1,nz0,ny0,nx0]
    movaps xmm3, [esi+16]      // xmm3 = [ny2,nx2,nz1,ny1]
    movaps xmm4, [esi+32]      // xmm4 = [nz3,ny3,nx3,nz2]
    prefetchnta    [esi+64]
    prefetchnta    [esi+96]

    movaps xmm0, xmm2       // xmm0 = [nx1,nz0,ny0,nx0]
    shufps xmm2, xmm3,73    // xmm2 = [nz1,ny1,nz0,ny0]
    shufps xmm3, xmm4,158   // xmm3 = [ny3,nx3,ny2,nx2]
    movaps xmm1, xmm2       // xmm1 = [nz1,ny1,nz0,ny0]
    shufps xmm1, xmm3, 216    // xmm1 = [ny3,ny2,ny1,ny0]
    shufps xmm0, xmm3, 140    // xmm0 = [nx3,nx2,nx1,nx0]
    shufps xmm2, xmm4,205   // xmm2 = [nz3,nz2,nz1,nz0]
    movaps    xmm4, xmm1    // xmm4 = [ny3,ny2,ny1,ny0]
```

```
  movaps    xmm3, xmm0     // xmm3 = [nx3,nx2,nx1,nx0]

  mulps     xmm0, [edx]         // xmm0 = [nx3*lx,nx2*lx,nx1*lx,nx0*lx]
  movaps    xmm5, xmm2     // xmm5 = [nz3,nz2,nz1,nz0]
  mulps     xmm1, [edx+16]   // xmm1 = [ny3*ly,ny2*ly,ny1*ly,ny0*ly]
  mulps     xmm2, [edx+32]   // xmm2 = [nz3*lz,nz2*lz,nz1*lz,nz0*lz]
  movaps    xmm7, [edx]      // xmm7 = [lx,lx,lx,lx]
  addps     xmm0, xmm1       // xmm0 = [nx3*lx+ny3*ly, ..,..,..]
  movaps    xmm1, [edx+16] // xmm1 = [ly,ly,ly,ly]
  addps     xmm0, xmm2       // xmm0 =
[costheta3,costheta2,costheta1,costheta0]
  movaps    xmm2, [edx+32] // xmm2 = [lz,lz,lz,lz]
  movaps    xmm6, xmm0     // xmm6 =
[costheta3,costheta2,costheta1,costheta0]

  addps     xmm0, xmm0     // xmm0 = [dt3,dt2,dt1,dt0], dti = 2.0*costhetai
  mulps     xmm3, xmm0     // xmm3 = [dt3*nx3,dt2*nx2,dt1*nx1,dt0*nx0]
  mulps     xmm4, xmm0     // xmm4 = [dt3*ny3,dt2*ny2,dt1*ny1,dt0*ny0]
  mulps     xmm5, xmm0     // xmm5 = [dt3*nz3,dt2*nz2,dt1*nz1,dt0*nz0]
  subps     xmm3, xmm7     // xmm3 = [Rx3,Rx2,Rx1,Rx0]
  subps     xmm4, xmm1     // xmm4 = [Ry3,Ry2,Ry1,Ry0]

  subps     xmm5, xmm2     // xmm5 = [Rz3,Rz2,Rz1,Rz0]

  mulps     xmm3, [edx+48] // xmm3 = [Rx3*Vx,Rx2*vx,Rx1*vx,Rx0*vx]
  xorps     xmm7, xmm7     // clear xmm7
  mulps     xmm4, [edx+64] // xmm4 = [Ry3*Vy,Ry2*vy,Ry1*vy,Ry0*vy]
  mulps     xmm5, [edx+80] // xmm5 = [Rz3*Vz,Rz2*vz,Rz1*vz,Rz0*vz]
  addps     xmm3, xmm4     // xmm3 = [Rx3*Vx+Ry3*Vy,..,..,..]
  subps     xmm7, xmm6     // xmm7 = [-costheta3,-costheta2,-costheta1,-
costheta0]
  addps     xmm5, xmm3     // xmm5 = [cosalpha3, cosalpha2, cosalpha1,
cosalpha0]
  maxps     xmm7, xmm6  // xmm7 = [Costheta3,Costheta2,Costheta1,Costheta0]

  movaps    xmm0,[edx+144] // xmm0 = [diffuseR,diffuseR,diffuseR,diffuseR]
  movaps    xmm1,[edx+160] // xmm1 = [diffuseG,diffuseG,diffuseG,diffuseG]
  movaps    xmm2,[edx+176] // xmm2 = [diffuseB,diffuseB,diffuseB,diffuseB]
  movaps    xmm3,[edx+96]    // xmm3 = [ambientR,ambientR,ambientR,ambientR]
  movaps    xmm4,[edx+112] // xmm4 = [ambientG,ambientG,ambientG,ambientG]
  movaps    xmm6,[edx+128] // xmm6 = [ambientB,ambientB,ambientB,ambientB]
  mulps     xmm0, xmm7     // xmm0 = [diffuseR*Costheta3,..,..,..]
  mulps     xmm1, xmm7     // xmm1 = [diffuseG*Costheta3,..,..,..]
  mulps     xmm2, xmm7     // xmm2 = [diffuseB*Costheta3,..,..,..]
  addps     xmm0,xmm3   // xmm0 = [diffuseR*Costheta3+ambientR,..,..,..]
```

```
   addps     xmm1,xmm4   // xmm1 = [diffuseG*Costheta3+ambientG,..,..,..]
   addps     xmm2,xmm6   // xmm2 = [diffuseB*Costheta3+ambientB,..,..,..]

   movaps  xmm3,four_255  // xmm3 = [255.0,255.0,255.0,255.0]
   xorps     xmm6, xmm6     // xmm6 = [0.0, 0.0, 0.0, 0.0]
   movaps  xmm7,[edx+192] // xmm7 = [specular,specular,specular,specular]

   cmpltps   xmm6,xmm5 //  0.0 < cosalpha?
   movmskps  eax, xmm6
   cmp       eax, 0
   je      Continue

   andps     xmm5,xmm6      // xmm5 = [cosa3,cosa2,cosa1,cosa0]
   mov       eax, phongconst
loop11:
   mulps     xmm7, xmm5       // xmm7 = [specular*cosalpha3,..,..,..]
   dec       eax
   jnz       loop11
   addps     xmm0, xmm7     // xmm0 = [fr3,fr2,fr1,fr0]
   addps     xmm1, xmm7     // xmm1 = [fg3,fg2,fg1,fg0]
   addps     xmm2, xmm7     // xmm2 = [fb3,fb2,fb1,fb0]

Continue:
   mulps       xmm0, xmm3     // xmm0 = [fR3,fR2,fR1,fR0]   , FRi = 255.0*fri
   mulps       xmm1, xmm3     // xmm1 = [fG3,fG2,fG1,fG0]
   mulps       xmm2, xmm3     // xmm2 = [fB3,fB2,fB1,fB0]
   movaps    xmm3, xmm0     // xmm3 = [fR3,fR2,fR1,fR0]
   shufps    xmm0, xmm1,136 // xmm0 = [fG2,fG0,fR2,fR0]  10001000
   shufps    xmm1, xmm2,221  // xmm1 = [fB3,fB1,fG3,fG1] 11011101
   shufps    xmm3, xmm2,141  // xmm3 = [fB2,fB0,fR3,fR1]  10001101
   movaps    xmm4, xmm1     // xmm4 = [fB3,fB1,fG3,fG1]
   shufps    xmm1, xmm0,216 // xmm1 = [fG2,fR2,fB1,fG1] 11011000
   shufps    xmm0, xmm3,40   // xmm0 = [fR1,fB0,fG0,fR0] 00101000
   shufps    xmm3, xmm4,215  // xmm3 = [fB3,fG3,fR3,fB2] 11010111

   cvtps2pi  mm0, xmm0      // mm0 = [G0,R0]
   shufps    xmm0,xmm0,14   // xmm0 = [*,*,fR1,fB0]
   cvtps2pi  mm1, xmm1      // mm1 = [B1,G1]
   shufps    xmm1,xmm1,14   // xmm1 = [*,*,fG2,fR2]
   cvtps2pi  mm2, xmm3      // mm2 = [R3,B2]
   shufps    xmm3,xmm3,14   // xmm3 = [*,*,fB3,fG3]
   cvtps2pi  mm3, xmm0      // mm3 = [R1,B0]
   cvtps2pi  mm4, xmm1      // mm4 = [G2,R2]
```

```
  cvtps2pi  mm5, xmm3      // mm5 = [B3,G3]

  packssdw  mm0,mm3     // mm0 = [R1,B0,G0,R0]
  packssdw  mm1,mm4     // mm4 = [G2,R2,B1,G1]
  packssdw  mm2,mm5     // mm2 = [B3,G3,R3,B2]
  pminsw    mm0, mm7      // mm0 =
[min(255,R1),min(255,B0),min(255,G0),min(255,R0)]
  pminsw    mm1, mm7      // mm1 =
[min(255,G2),min(255,R2),min(255,B1),min(255,G1)]
  pminsw    mm2, mm7      // mm2 =
[min(255,B3),min(255,G3),min(255,R3),min(255,B3)]

  packuswb  mm0, mm1      // mm0 = [G2,R2,B1,G1,R1,B0,G0,R0]
  movq      [edi], mm0
  packuswb  mm2, mm2      // mm2 = [*,*,*,*,B3,G3,R3,B2]
  movd      [edi+8], mm2

  add       esi, 48     // Next four Vectex
  add       edi, 12

  dec     ecx
  jne     NextFourVectex

DoRemainder:
  mov     ecx, reNum
  cmp     ecx, 0
  je   fininsh

  // Data set up for the rest of vertices.
  push edx
  mov     eax, 0FFFFFFh
  movd mm2, eax
  mov     eax, View
  mov     edx, light
  movups xmm0, [eax]
  movups xmm4, [edx]    // xmm4 = [*,lz,ly,lx]
  mov     eax, diffuse
  mov     edx, ambient
  movups xmm3, [eax]    // xmm3 = [*,diffuse->B,diffuse->G,diffuse->R]
  movups xmm2, [edx]    // xmm2 = [*,ambient->B,ambient->G,ambient->R]
  pop     edx
  movaps [edx], xmm0    // save view in a memory buffer

NextVectex:
```

```
    movups xmm0, [esi]    // xmm0 = [*,nz,ny,nx]
    movaps xmm1, xmm0  // xmm1 = [*,nz,ny,nx]
    mulps  xmm0, xmm4     // xmm0 = [*, lz*nz, ly*ny, lx*nx]

    movaps xmm6, xmm0     // xmm6 = [*, lz*nz, ly*ny, lx*nx]
    shufps xmm0, xmm0,9      // xmm0 = [*, lx*nx, lz*nz, ly*ny]
    addps  xmm0, xmm6        // xmm0 = [*,
lx*nx+lz*nz,lz*nz+ly*ny,lx*nx+ly*ny]
    shufps xmm6, xmm6,18  // xmm6 = [*, ly*ny, lx*nx, lz*nz]
    addps  xmm0, xmm6        // xmm0 = [*, cosTheta, cosTheta, cosTheta]
    movaps xmm7, xmm0     // xmm7 = [*, cosTheta, cosTheta, cosTheta]

    // Calculate the mirror vector: R = 2*(dot(N,L)N - L
    addps  xmm0, xmm0     // xmm0 = [*,dt,dt,dt], dt = 2*cosTheta
    mulps  xmm0, xmm1     // xmm0 = [*,dt*nz,dt*ny,dt*nx]
    subps  xmm0, xmm4     // xmm0 = [*,dt*nz-lz,dt*ny-ly,dt*nx-lx]

    // calculate cosAlpha = cos(Alpha), where
    // Alpha is the angle between view vector V and mirror vector R
    mulps  xmm0, [edx]       // xmm0 = [*, Rz*Vz,Ry*Vy,Rx*Vx]

    movaps xmm6, xmm0     // xmm6 = [*, Rz*Vz,Ry*Vy,Rx*Vx]
    shufps xmm0, xmm0,9      //  xmm0 = [*, Rx*Vx, Rz*Vz,Ry*Vy]
    addps  xmm0, xmm6        // xmm0 = [*, Rx*Vx + Rz*Vz,
Rz*Vz+Ry*Vy,Ry*Vy+Rx*Vx]
    shufps xmm6, xmm6,18  // xmm6 = [*, Ry*Vy,Rx*Vx,Rz*Vz]
    addps  xmm0, xmm6        //  xmm0 = [*, cosAlpha,cosAlpha,cosAlpha]

    // If cosTheta < 0.0, the point is on back side of the surface,
    xorps  xmm5, xmm5        //  xmm5 = [0.0, 0.0, 0.0, 0.0]
    comiss xmm7, xmm5     // cosTheta < 0.0?  (If True CF = 1)
    jnc    Continue2

    subps  xmm5, xmm7        // xmm5 = [-cosTheta,-cosTheta,-cosTheta,-
cosTheta]
    movaps xmm7,xmm5   // xmm7 = [-cosTheta,-cosTheta,-cosTheta,-cosTheta]

Continue2:
    mulps  xmm7, xmm3        // xmm7 = [*,cosTheta*diffuseB,..,..,..]
    addps  xmm7, xmm2        // xmm7 = Ambient + Diffuse
    xorps  xmm6, xmm6        // clear xmm6
    comiss xmm0, xmm6     // cosAlpha > 0.0 ? (If cosAlpha > 0.0, CF = 0)
    jc   DoCnvert         //  If 0.0 > cosAlpha, no Specular components.
    mov    edx, phongconst
```

```
   movss   xmm6, specular   // xmm6 = [0.0,0.0,0.0,specular]

loop21:
   mulss   xmm6, xmm0       // xmm6 = [*,*,*, cosn*cosAlpha]
   dec     edx
   jnz     loop21

   shufps  xmm6,xmm6,0    // xmm6 = [*,spN,spN,spN]
   addps   xmm7, xmm6       // xmm7 = [*, fB, fG, fR]

DoCnvert:
   mulps   xmm7, four_255   // xmm7 = [*, 255*fB, 255*fG, 255*fR]

   // Convert (fR,fG,fB) from packed floats to packed bytes (R,G,B)
   // and write only three bytes memory location specified by [edi]
   cvtps2pi  mm0, xmm7   // mm0 = [G,R]
   shufps  xmm7, xmm7,2     // xmm1 = [*,*,*,B]
   cvtps2pi  mm1, xmm7   // mm0 = [*,B]
   packssdw  mm0,mm1   // mm0 = [*,B,G,R]
   pminsw    mm0,mm7
   packuswb  mm0,mm0   // mm0 = [*,*,*,*,*,B,G,R]
   // write three bytes output of mm0 to memory [edi]
   maskmovq  mm0, mm2

   add       esi, 12
   add       edi, 3
   dec  ecx
   jne  NextVectex

finish:

   emms
   }

}/* Intensitys_AOS_XMM() */
```

7 Streaming SIMD Extensions SoA Assembly Code Example

The following sections provide code samples for both an unoptimized and an optimized version of the Phong equation algorithm using the SoA data layout.

7.1 SoA Unoptimized Version

```
void _Intensitys_SOA_XMM(int num_vertex,
            float *normx,
            float *normy,
            float *normz,
            Point_3d *light,
            Point_3d *View,
            F_RGB *ambient,
            F_RGB *diffuse,
            float   specular,
            long phongconst,
            struct RGB_color *rgb)

{
  float buffer[56];
  float *tem;
  long _rgb;
  int reNum;

  tem = (float *) (((unsigned)buffer + 15) & ~15);
__asm{
  // Data Setup
  mov       esi, light
  mov       edi, View
  mov       edx, tem
  mov       ecx, ambient

  mov       eax,000FF00FFh
  movd      mm7, eax       // mm7 = [0,0,255,255]
  punpcklwd mm7, mm7    // mm7 = [255,255,255,255]
  mov       eax, diffuse
  movups    xmm0, [esi]    // xmm0 = [*,lz,ly,lx]
  movaps    xmm1, xmm0     // xmm1 = [*,lz,ly,lx]
  movaps    xmm2, xmm0     // xmm2 = [*,lz,ly,lx]

  shufps    xmm0, xmm0, 0    // xm0 = [lx,lx,lx,lx]
  shufps    xmm1, xmm1, 85   // xm1 = [ly,ly,ly,ly]
  shufps    xmm2, xmm2, 170  // xm2 = [lz,lz,lz,lz]

  movaps    [edx], xmm0
  movaps    [edx+16], xmm1
```

```
movaps    [edx+32], xmm2

movups    xmm3, [edi]    // xmm3 = [*,vz,vy,vx]
movaps    xmm4, xmm3     // xmm4 = [*,vz,vy,vx]
movaps    xmm5, xmm3     // xmm5 = [*,vz,vy,vx]

shufps    xmm3, xmm3, 0    // xmm3 = [vx,vx,vx,vx]
shufps    xmm4, xmm4, 85   // xmm4 = [vy,vy,vy,vy]
shufps    xmm5, xmm5, 170  // xmm5 = [vz,vz,vz,vz]

movaps    [edx+48], xmm3
movaps    [edx+64], xmm4
movaps    [edx+80], xmm5

movups    xmm0, [ecx]    // xmm0 = [*,ambientB,ambientG,ambientR]
movaps    xmm1, xmm0     // xmm1 = [*,ambientB,ambientG,ambientR]
movaps    xmm2, xmm0     // xmm2 = [*,ambientB,ambientG,ambientR]

shufps    xmm0, xmm0, 0    // xmm0 = [ambientR,ambientR,ambientR,ambientR]
shufps    xmm1, xmm1, 85   // xmm1 = [ambientG,ambientG,ambientG,ambientG]
shufps    xmm2, xmm2, 170  // xmm2 = [ambientB,ambientB,ambientB,ambientB]


movaps    [edx+96], xmm0
movaps    [edx+112], xmm1
movaps    [edx+128], xmm2

movups    xmm3, [eax]    // xmm3 = [*,diffuseB,diffuseG,diffuseR]
movaps    xmm4, xmm3     // xmm4 = [*,diffuseB,diffuseG,diffuseR]
movaps    xmm5, xmm3     // xmm5 = [*,diffuseB,diffuseG,diffuseR]

shufps    xmm3, xmm3, 0    // xmm3 = [diffuseR,diffuseR,diffuseR,diffuseR]
shufps    xmm4, xmm4, 85   // xmm4 = [diffuseG,diffuseG,diffuseG,diffuseG]
shufps    xmm5, xmm5, 170  // xmm5 = [diffuseB,diffuseB,diffuseB,diffuseB]

movaps    [edx+144],xmm3 // [diffuseR,diffuseR,diffuseR,diffuseR]
movaps    [edx+160],xmm4 // [diffuseG,diffuseG,diffuseG,diffuseG]
movaps    [edx+176],xmm5 // [diffuseB,diffuseB,diffuseB,diffuseB]

movss     xmm6, specular   // xmm6 = [0.0,0.0,0.0,specular]
shufps    xmm6, xmm6, 0    // xmm6 = [specular,specular,specular,specular]
movaps    [edx+192], xmm6// [specular,specular,specular,specular]
```

```
  mov       esi, rgb
  mov       _rgb, esi
  mov       ecx, num_vertex
  mov       eax, ecx
  shr         ecx, 2        // ecx = num_vertex/4
  mov       esi, ecx
  shl         esi, 2        // esi = 4*(num_vertex/4)
  sub         eax, esi      // number of vertices remains
  mov       reNum, eax

  mov       esi, normx
  mov       eax, normy
  mov       edi, normz

  cmp       ecx, 0
  je      DoRemainder

NextFourVertex:
  movaps    xmm0, [esi]    // xmm0 = [nx3,nx2,nx1,nx0]
  movaps    xmm1, [eax]    // xmm1 = [ny3,ny2,ny1,ny0]
  movaps    xmm2, [edi]    // xmm2 = [nz3,nz2,nz1,nz0]
  movaps    xmm3, xmm0     // xmm3 = [nx3,nx2,nx1,nx0]
  movaps    xmm4, xmm1     // xmm4 = [ny3,ny2,ny1,ny0]
  movaps    xmm5, xmm2     // xmm5 = [nz3,nz2,nz1,nz0]

  // costheta = dot(norm,ldir)
  mulps     xmm0, [edx]      // xmm0 = [nx3*lx,nx2*lx,nx1*lx,nx0*lx]
  mulps     xmm1, [edx+16]   // xmm1 = [ny3*ly,ny2*ly,ny1*ly,ny0*ly]
  mulps     xmm2, [edx+32]   // xmm2 = [nz3*lz,nz2*lz,nz1*lz,nz0*lz]
  addps     xmm0, xmm1       // xmm0 = [nx3*lx+ny3*ly, ..,..,..]
  addps     xmm0, xmm2       // xmm0 =
[costheta3,costheta2,costheta1,costheta0]

  movaps    xmm6, xmm0     // xmm6 =
[costheta3,costheta2,costheta1,costheta0]

  // calculate the reflection vector R
  addps     xmm0, xmm0       // xmm0 = [dt3,dt2,dt1,dt0], dti =
2.0*costhetai
  mulps     xmm3, xmm0       // xmm3 = [dt3*nx3,dt2*nx2,dt1*nx1,dt0*nx0]
  mulps     xmm4, xmm0       // xmm4 = [dt3*ny3,dt2*ny2,dt1*ny1,dt0*ny0]
  mulps     xmm5, xmm0       // xmm5 = [dt3*nz3,dt2*nz2,dt1*nz1,dt0*nz0]
  subps     xmm3, [edx]      // xmm3 = [Rx3,Rx2,Rx1,Rx0]
```

```
  subps     xmm4, [edx+16]   // xmm4 = [Ry3,Ry2,Ry1,Ry0]

  subps     xmm5, [edx+32]   // xmm5 = [Rz3,Rz2,Rz1,Rz0]


  // cosalpha = dot(R,v_pnt)
  mulps     xmm3, [edx+48]   // xmm3 = [Rx3*Vx,Rx2*vx,Rx1*vx,Rx0*vx]
  mulps     xmm4, [edx+64]   // xmm4 = [Ry3*Vy,Ry2*vy,Ry1*vy,Ry0*vy]
  mulps     xmm5, [edx+80]   // xmm5 = [Rz3*Vz,Rz2*vz,Rz1*vz,Rz0*vz]
  addps     xmm3, xmm4       // xmm3 = [Rx3*Vx+Ry3*Vy,..,..,..]
  addps     xmm5, xmm3       // xmm5 = [cosalpha3, cosalpha2, cosalpha1,
cosalpha0]

  // set Costheta = |costheta|
  xorps     xmm7, xmm7       // clear  xmm7
  subps     xmm7, xmm6       // xmm7 = [-costheta3,-costheta2,-costheta1,-
costheta0]
  maxps     xmm7, xmm6       // xmm7 =
[Costheta3,Costheta2,Costheta1,Costheta0]

  movaps    xmm0,[edx+144]// xmm0 = [diffuseR,diffuseR,diffuseR,diffuseR]
  movaps    xmm1,[edx+160]// xmm1 = [diffuseG,diffuseG,diffuseG,diffuseG]
  movaps    xmm2,[edx+176]// xmm2 = [diffuseB,diffuseB,diffuseB,diffuseB]
  movaps    xmm3,[edx+96] // xmm3 = [ambientR,ambientR,ambientR,ambientR]
  movaps    xmm4,[edx+112]// xmm4 = [ambientG,ambientG,ambientG,ambientG]
  movaps    xmm6,[edx+128]// xmm6 = [ambientB,ambientB,ambientB,ambientB]
  mulps     xmm0, xmm7       // xmm0 = [diffuseR*Costheta3,..,..,..]
  mulps     xmm1, xmm7       // xmm1 = [diffuseG*Costheta3,..,..,..]
  mulps     xmm2, xmm7       // xmm2 = [diffuseB*Costheta3,..,..,..]
  addps     xmm0,xmm3      // xmm0 = [diffuseR*Costheta3+ambientR,..,..,..]
  addps     xmm1,xmm4      // xmm1 = [diffuseG*Costheta3+ambientG,..,..,..]
  addps     xmm2,xmm6      // xmm2 = [diffuseB*Costheta3+ambientB,..,..,..]

  push   edi
  xorps     xmm6, xmm6       // xmm6 = [0.0, 0.0, 0.0, 0.0]
  movaps    xmm7,[edx+192]// xmm7 = [specular,specular,specular,specular]
  cmpltps   xmm6,xmm5   // 0.0 < cosalpha3
  movmskps  edi, xmm6
  cmp       edi, 0
  je     Continue1
  andps     xmm5,xmm6      // xmm5 = [cosa3,cosa2,cosa1,cosa0], cosai = (0.0
< cosai)? cosai:0.0
  mov       edi, phongconst

loop11:
  mulps  xmm7, xmm5          // xmm7 = [specular*cosalpha3,..,..,..]
```

```
   dec     edi
   jnz     loop11

   addps     xmm0, xmm7      // xmm0 = [fr3,fr2,fr1,fr0]

   addps     xmm1, xmm7      // xmm1 = [fg3,fg2,fg1,fg0]

   addps     xmm2, xmm7      // xmm2 = [fb3,fb2,fb1,fb0]

Continue1:
   mulps     xmm0, four_255   // xmm0 = [fR3,fR2,fR1,fR0]
   mulps     xmm1, four_255   // xmm1 = [fG3,fG2,fG1,fG0]
   mulps     xmm2, four_255   // xmm2 = [fB3,fB2,fB1,fB0]

   // Interleave R,G,B
   movaps    xmm3, xmm0     // xmm3 = [fR3,fR2,fR1,fR0]
   shufps    xmm0, xmm1,136 // xmm0 = [fG2,fG0,fR2,fR0]
   shufps    xmm3, xmm2,141   // xmm3 = [fB2,fB0,fR3,fR1]
   shufps    xmm1, xmm2,221   // xmm1 = [fB3,fB1,fG3,fG1]
   movaps xmm4, xmm1        // xmm4 = [fB3,fB1,fG3,fG1]
   shufps    xmm1, xmm0,216 // xmm1 = [fG2,fR2,fB1,fG1]
   shufps    xmm0, xmm3,40    // xmm0 = [fR1,fB0,fG0,fR0]
   shufps    xmm3, xmm4,215   // xmm3 = [fB3,fG3,fR3,fB2]
   cvtps2pi  mm0, xmm0      // mm0 = [G0,R0]
   cvtps2pi  mm1, xmm1      // mm1 = [B1,G1]
   cvtps2pi  mm2, xmm3      // mm2 = [R3,B2]

   shufps    xmm0,xmm0,14     // xmm0 = [*,*,fR1,fB0]
   shufps    xmm1,xmm1,14     // xmm1 = [*,*,fG2,fR2]
   shufps    xmm3,xmm3,14     // xmm3 = [*,*,fB3,fG3]
   cvtps2pi  mm3, xmm0      // mm3 = [R1,B0]
   cvtps2pi  mm4, xmm1      // mm4 = [G2,R2]
   cvtps2pi  mm5, xmm3      // mm5 = [B3,G3]

   packssdw  mm0,mm3     // mm0 = [R1,B0,G0,R0]
   packssdw  mm1,mm4     // mm4 = [G2,R2,B1,G1]
   packssdw  mm2,mm5     // mm2 = [B3,G3,R3,B2]
   pminsw    mm0, mm7    // mm0 =
[min(255,R1),min(255,B0),min(255,G0),min(255,R0)]
   pminsw    mm1, mm7    // mm1 =
[min(255,G2),min(255,R2),min(255,B1),min(255,G1)]
   pminsw    mm2, mm7    // mm2 =
[min(255,B3),min(255,G3),min(255,R3),min(255,B3)]
   packuswb  mm0, mm1       // mm0 = [G2,R2,B1,G1,R1,B0,G0,R0]
   packuswb  mm2, mm2       // mm2 = [*,*,*,*,B3,G3,R3,B2]
```

```
  mov       edi, _rgb
  movq    [edi], mm0
  movd    [edi+8], mm2
  add       edi, 12
  mov       _rgb, edi
  pop       edi
  add       esi, 16
  add       eax, 16
  add       edi, 16

  dec       ecx
  jnz       NextFourVertex

  // Data set up for the rest of vertices.
  DoRemainder:
  mov     ecx, reNum
  cmp     ecx, 0
  je   fininsh

  // Data set up for the rest of vertices.
  push edx
  mov     edx, 0FFFFFFh
  movd mm4, edx
  pop     edx

NextVertex:
  // one vertex for each iteration
  movss     xmm0, [esi]    // xmm0 = [*,*,*,nx]
  movss     xmm1, [eax]    // xmm1 = [*,*,*,ny]
  movss     xmm2, [edi]    // xmm2 = [*,*,*,nz]
  movss     xmm3, xmm0     // xmm3 = [*,*,*,nx]
  movss     xmm4, xmm1     // xmm4 = [*,*,*,ny]
  movss     xmm5, xmm2     // xmm5 = [*,*,*,nz]

  // costheta = dot(norm,ldir)
  mulss     xmm0, [edx]    // xmm0 = [*,*,*,nx*lx]
  mulss     xmm1, [edx+16] // xmm1 = [*,*,*,ny*ly]
  mulss     xmm2, [edx+32] // xmm2 = [*,*,*,nz*lz]
  addss     xmm0, xmm1     // xmm0 = [*,*,*,nx*lx+ny*ly]
  addss     xmm0, xmm2     // xmm0 = [*,*,*,costheta]

  movss     xmm6, xmm0     // xmm6 = [costheta]
```

```
    // calculate the reflection vector R
    addss     xmm0, xmm0     // xmm0 = [*,*,*,dt], dt = 2.0*costheta
    mulss     xmm3, xmm0     // xmm3 = [*,*,*,dt*nx]
    mulss     xmm4, xmm0     // xmm4 = [*,*,*,dt*ny]
    mulss     xmm5, xmm0     // xmm5 = [*,*,*,dt*nz]
    subss     xmm3, [edx]    // xmm3 = [*,*,*,dt*nx-lx]

    subss     xmm4, [edx+16] // xmm4 = [*,*,*,dt*ny-ly]

    subss     xmm5, [edx+32] // xmm5 = [*,*,*,dt*nz-lz]

    // cosalpha = dot(R,v_pnt)
    mulss     xmm3, [edx+48] // xmm3 = [*,*,*,Rx*vx]
    mulss     xmm4, [edx+64] // xmm4 = [*,*,*,Ry*vy]
    mulss     xmm5, [edx+80] // xmm5 = [*,*,*,Rz*vz]
    addss     xmm3, xmm4     // xmm3 = [*,*,*,Rx*Vx+Ry*Vy]
    addss     xmm5, xmm3     // xmm5 = [*,*,*,cosalpha]

    xorps     xmm7, xmm7     // clear xmm7
    subss     xmm7, xmm6     // xmm7 = [*,*,*,-costheta]
    maxss     xmm7, xmm6     // xmm7 = [*,*,*,Costheta], Costheta = |costheta|

    movss     xmm0,[edx+144] // xmm0 = [*,*,*,diffuseR]
    movss     xmm1,[edx+160] // xmm1 = [*,*,*,diffuseG]
    movss     xmm2,[edx+176] // xmm2 = [*,*,*,diffuseB]
    movss     xmm3,[edx+96]  // xmm3 = [*,*,*,ambientR]
    movss     xmm4,[edx+112] // xmm4 = [*,*,*,ambientG]
    movss     xmm6,[edx+128] // xmm6 = [*,*,*,ambientB]

    mulps     xmm0, xmm7     // xmm0 = [*,*,*,diffuseR*Costheta]
    mulps     xmm1, xmm7     // xmm1 = [*,*,*,diffuseG*Costheta]
    mulps     xmm2, xmm7     // xmm2 = [*,*,*,diffuseB*Costheta]
    addps     xmm0,xmm3   // xmm0 = [*,*,*,diffuseR*Costheta3+ambientR]
    addps     xmm1,xmm4   // xmm1 = [*,*,*,diffuseG*Costheta3+ambientG]
    addps     xmm2,xmm6   // xmm2 = [*,*,*,diffuseB*Costheta3+ambientB]

    xorps     xmm6, xmm6     // xmm6 = [0.0, 0.0, 0.0, 0.0]
    push    edi
    comiss    xmm5, xmm6     // check if cosAlpha < 0.0 ? (cosAlpha < 0.0 ==>
  CF = 1)
    jc      Continue2

    movss     xmm7,[edx+192] // xmm7 = [*,*,*,specular]
```

```
   mov       edi, phongconst

loop21:
   mulss   xmm7, xmm5       // xmm7 = [specular*cosalpha3,..,..,..]
   dec     edi
   jnz     loop21

   addss     xmm0, xmm7     // xmm0 = [*,*,*,fr]

   addss     xmm1, xmm7     // xmm1 = [*,*,*,fg]

   addss     xmm2, xmm7     // xmm2 = [*,*,*,fb]

Continue2:
   mulss     xmm0, four_255 // xmm0 = [*,*,*,fR]
   mulss     xmm1, four_255 // xmm1 = [*,*,*,fG]
   mulss     xmm2, four_255 // xmm2 = [*,*,*,fB]

   // Interleave R,G,B and write to memory
   mov       edi, _rgb
   cvtps2pi  mm0, xmm0   // mm0 = [*,R]
   cvtps2pi  mm1, xmm1   // mm1 = [*,G]
   cvtps2pi  mm2, xmm2   // mm2 = [*,B]
   punpckldq mm0, mm2    // mm0 = [B,R]
   packssdw  mm0, mm0    // mm0 = [*,*,B,R]
   punpcklwd mm0, mm1  // mm0 = [*,B,G,R]
   pminsw    mm0, mm7  // mm0 = [*,min(255,B),min(255,G),min(255,R)]
   packuswb  mm0, mm0    // mm0 = [*,*,*,*,*,B,G,R]
   // write three bytes R,G,B in mm0 to memory [edi]
   maskmovq  mm0, mm4

   add       edi, 3
   mov       _rgb, edi
   pop       edi
   add       esi, 4
   add       eax, 4
   add       edi, 4
   dec          ecx
   jnz          NextVertex
fininsh:

   emms

   }
```

```
}
```

7.2 SoA Optimized Version

```
void Intensitys_SOA_XMM(int num_vertex,
            float *normx,
            float *normy,
            float *normz,
            Point_3d *light,
            Point_3d *View,
            F_RGB *ambient,
            F_RGB *diffuse,
            float   specular,
            long phongconst,
            struct RGB_color *rgb)

{
  float buffer[56];
  float *tem;
  long _rgb;
  int reNum;
  tem = (float *) (((unsigned)buffer + 15) & ~15);

__asm{
  // Data Setup
  mov       eax,000FF00FFh
  mov       esi, light
  mov       edi, View
  mov       edx, tem
  movd      mm7, eax       // mm7 = [0,0,255,255]
  movups    xmm0, [esi]    // xmm0 = [*,lz,ly,lx]
  mov       ecx, ambient
  mov       eax, diffuse
  punpcklwd mm7, mm7    // mm7 = [255,255,255,255]
  movaps    xmm1, xmm0     // xmm1 = [*,lz,ly,lx]
  movaps    xmm2, xmm0     // xmm2 = [*,lz,ly,lx]
  shufps    xmm0, xmm0, 0    // xm0 = [lx,lx,lx,lx]
  movups    xmm3, [edi]    // xmm3 = [*,vz,vy,vx]
  shufps    xmm1, xmm1, 85   // xm1 = [ly,ly,ly,ly]
  shufps    xmm2, xmm2, 170  // xm2 = [lz,lz,lz,lz]

  movaps    [edx], xmm0
  movaps    [edx+16], xmm1
```

```
movaps    [edx+32], xmm2
movaps    xmm4, xmm3     // xmm4 = [*,vz,vy,vx]
movaps    xmm5, xmm3     // xmm5 = [*,vz,vy,vx]

shufps    xmm3, xmm3, 0    // xmm3 = [vx,vx,vx,vx]
movups    xmm0, [ecx]    // xmm0 = [*,ambientB,ambientG,ambientR]
shufps    xmm4, xmm4, 85   // xmm4 = [vy,vy,vy,vy]
shufps    xmm5, xmm5, 170  // xmm5 = [vz,vz,vz,vz]

movaps    [edx+48], xmm3
movaps    [edx+64], xmm4
movaps    [edx+80], xmm5
movaps    xmm1, xmm0     // xmm1 = [*,ambientB,ambientG,ambientR]
movaps    xmm2, xmm0     // xmm2 = [*,ambientB,ambientG,ambientR]
shufps    xmm0, xmm0, 0    // xmm0 = [ambientR,ambientR,ambientR,ambientR]
movups    xmm3, [eax]    // xmm3 = [*,diffuseB,diffuseG,diffuseR]
shufps    xmm1, xmm1, 85   // xmm1 = [ambientG,ambientG,ambientG,ambientG]
shufps    xmm2, xmm2, 170  // xmm2 = [ambientB,ambientB,ambientB,ambientB]

movaps    [edx+96], xmm0
movaps    [edx+112], xmm1
movaps    [edx+128], xmm2

movaps    xmm4, xmm3     // xmm4 = [*,diffuseB,diffuseG,diffuseR]
movaps    xmm5, xmm3     // xmm5 = [*,diffuseB,diffuseG,diffuseR]
shufps    xmm3, xmm3, 0    // xmm3 = [diffuseR,diffuseR,diffuseR,diffuseR]
movss     xmm6, specular   // xmm6 = [0.0,0.0,0.0,specular]
shufps    xmm4, xmm4, 85   // xmm4 = [diffuseG,diffuseG,diffuseG,diffuseG]
shufps    xmm5, xmm5, 170  // xmm5 = [diffuseB,diffuseB,diffuseB,diffuseB]
movaps    [edx+144],xmm3 // [diffuseR,diffuseR,diffuseR,diffuseR]
movaps    [edx+160],xmm4 // [diffuseG,diffuseG,diffuseG,diffuseG]
movaps    [edx+176],xmm5 // [diffuseB,diffuseB,diffuseB,diffuseB]
shufps    xmm6, xmm6, 0    // xmm6 = [specular,specular,specular,specular]
mov       esi, rgb
mov       _rgb, esi
mov       ecx, num_vertex
movaps    [edx+192], xmm6    // [specular,specular,specular,specular]

mov       eax, ecx
shr         ecx, 2             // ecx = num_vertex/4
mov       esi, ecx
shl         esi, 2             // esi = 4*(num_vertex/4)
sub         eax, esi           // number of vertices remains
```

```
   mov       reNum, eax

   mov       esi, normx
   mov       eax, normy
   mov       edi, normz
   cmp       ecx, 0
   je      DoRemainder

NextFourVectex:
   movaps    xmm0, [esi]      // xmm0 = [nx3,nx2,nx1,nx0]
   movaps    xmm1, [eax]      // xmm1 = [ny3,ny2,ny1,ny0]
   movaps    xmm2, [edi]      // xmm2 = [nz3,nz2,nz1,nz0]
   prefetchnta    [esi+64]
   prefetchnta    [eax+64]
   prefetchnta    [edi+64]
   movaps    xmm3, xmm0    // xmm3 = [nx3,nx2,nx1,nx0]
   movaps    xmm4, xmm1    // xmm4 = [ny3,ny2,ny1,ny0]
   movaps    xmm5, xmm2    // xmm5 = [nz3,nz2,nz1,nz0]
   mulps     xmm0, [edx]      // xmm0 = [nx3*lx,nx2*lx,nx1*lx,nx0*lx]
   mulps     xmm1, [edx+16]   // xmm1 = [ny3*ly,ny2*ly,ny1*ly,ny0*ly]
   mulps     xmm2, [edx+32]   // xmm2 = [nz3*lz,nz2*lz,nz1*lz,nz0*lz]
   addps     xmm0, xmm1       // xmm0 = [nx3*lx+ny3*ly, ..,..,..]
   movaps    xmm7, [edx]   // xmm7 = [lx,lx,lx,lx]
   addps     xmm0, xmm2       // xmm0 =
[costheta3,costheta2,costheta1,costheta0]
   movaps    xmm6, [edx+16]// xmm6 = [ly,ly,ly,ly]
   movaps    xmm1, xmm0    // xmm1 =
[costheta3,costheta2,costheta1,costheta0]
   addps     xmm0, xmm0       // xmm0 = [dt3,dt2,dt1,dt0], dti =
2.0*costhetai
   movaps    xmm2, [edx+32]// xmm2 = [lz,lz,lz,lz]

   mulps     xmm3, xmm0    // xmm3 = [dt3*nx3,dt2*nx2,dt1*nx1,dt0*nx0]
   mulps     xmm4, xmm0    // xmm4 = [dt3*ny3,dt2*ny2,dt1*ny1,dt0*ny0]
   mulps     xmm5, xmm0    // xmm5 = [dt3*nz3,dt2*nz2,dt1*nz1,dt0*nz0]
   subps     xmm3, xmm7    // xmm3 = [Rx3,Rx2,Rx1,Rx0]

   subps     xmm4, xmm6    // xmm4 = [Ry3,Ry2,Ry1,Ry0]

   subps     xmm5, xmm2    // xmm5 = [Rz3,Rz2,Rz1,Rz0]

   mulps     xmm3, [edx+48]// xmm3 = [Rx3*Vx,Rx2*vx,Rx1*vx,Rx0*vx]
   xorps     xmm7, xmm7    // clear xmm7
   mulps     xmm4, [edx+64]// xmm4 = [Ry3*Vy,Ry2*vy,Ry1*vy,Ry0*vy]
   mulps     xmm5, [edx+80]// xmm5 = [Rz3*Vz,Rz2*vz,Rz1*vz,Rz0*vz]
```

```
  subps     xmm7, xmm1     // xmm7 = [-costheta3,-costheta2,-costheta1,-
costheta0]
  addps     xmm3, xmm4     // xmm3 = [Rx3*Vx+Ry3*Vy,..,..,..]

  maxps       xmm7, xmm1     //] Costheta = |costheta|
  addps       xmm5, xmm3     // xmm5 = [cosalpha3, cosalpha2, cosalpha1,
cosalpha0]
  movaps    xmm0,[edx+144] // xmm0 = [diffuseR,diffuseR,diffuseR,diffuseR]
  movaps    xmm1,[edx+160] // xmm1 = [diffuseG,diffuseG,diffuseG,diffuseG]
  movaps    xmm2,[edx+176] // xmm2 = [diffuseB,diffuseB,diffuseB,diffuseB]
  movaps    xmm3,[edx+96]  // xmm3 = [ambientR,ambientR,ambientR,ambientR]
  movaps    xmm4,[edx+112] // xmm4 = [ambientG,ambientG,ambientG,ambientG]
  movaps    xmm6,[edx+128] // xmm6 = [ambientB,ambientB,ambientB,ambientB]

  mulps     xmm0, xmm7     // xmm0 = [diffuseR*Costheta3,..,..,..]
  mulps     xmm1, xmm7     // xmm1 = [diffuseG*Costheta3,..,..,..]
  mulps     xmm2, xmm7     // xmm2 = [diffuseB*Costheta3,..,..,..]
  addps     xmm0,xmm3    // xmm0 = [diffuseR*Costheta3+ambientR,..,..,..]
  addps     xmm1,xmm4    // xmm1 = [diffuseG*Costheta3+ambientG,..,..,..]
  addps     xmm2,xmm6    // xmm2 = [diffuseB*Costheta3+ambientB,..,..,..]
  movaps  xmm3,four_255  // xmm3 = [255.0,255.0,255.0,255.0]

  push      edi
  xorps       xmm6, xmm6       // xmm6 = [0.0, 0.0, 0.0, 0.0]
  movaps    xmm7,[edx+192]   // xmm7 = [specular,specular,specular,specular]
  cmpltps   xmm6,xmm5      // 0.0 < cosalpha?
  movmskps  edi, xmm6
  cmp       edi, 0
  je        Continue1
  andps       xmm5,xmm6      // xmm5 = [cosa3,cosa2,cosa1,cosa0]
  mov       edi, phongconst

loop11:
  mulps   xmm7, xmm5       // xmm7 = [specular*cosalpha3,..,..,..]
  dec     edi
  jnz     loop11

  addps     xmm0, xmm7     // xmm0 = [fr3,fr2,fr1,fr0]

  addps     xmm1, xmm7     // xmm1 = [fg3,fg2,fg1,fg0]

  addps     xmm2, xmm7     // xmm2 = [fb3,fb2,fb1,fb0]

Continue1:
```

```
mulps     xmm0, xmm3     // xmm0 = [fR3,fR2,fR1,fR0]
mulps     xmm1, xmm3     // xmm1 = [fG3,fG2,fG1,fG0]
mulps     xmm2, xmm3     // xmm2 = [fB3,fB2,fB1,fB0]
movaps xmm3, xmm0     // xmm3 = [fR3,fR2,fR1,fR0]
shufps    xmm0, xmm1,136 // xmm0 = [fG2,fG0,fR2,fR0]  10001000
shufps    xmm1, xmm2,221   // xmm1 = [fB3,fB1,fG3,fG1] 11011101
shufps    xmm3, xmm2,141   // xmm3 = [fB2,fB0,fR3,fR1]  10001101
movaps    xmm4, xmm1     // xmm4 = [fB3,fB1,fG3,fG1]
shufps    xmm1, xmm0,216 // xmm1 = [fG2,fR2,fB1,fG1] 11011000
shufps    xmm0, xmm3,40    // xmm0 = [fR1,fB0,fG0,fR0] 00101000
shufps    xmm3, xmm4,215    // xmm3 = [fB3,fG3,fR3,fB2] 11010111

cvtps2pi  mm1, xmm1   // mm1 = [B1,G1]
cvtps2pi  mm0, xmm0   // mm0 = [G0,R0]
cvtps2pi  mm2, xmm3   // mm2 = [R3,B2]
shufps    xmm1,xmm1,14  // xmm1 = [*,*,fG2,fR2]
shufps    xmm0,xmm0,14  // xmm0 = [*,*,fR1,fB0]
shufps    xmm3,xmm3,14  // xmm3 = [*,*,fB3,fG3]
mov       edi, _rgb
cvtps2pi  mm4, xmm1   // mm4 = [G2,R2]
cvtps2pi  mm3, xmm0   // mm3 = [R1,B0]
cvtps2pi  mm5, xmm3   // mm5 = [B3,G3]
add       edi, 12
packssdw  mm0,mm3     // mm0 = [R1,B0,G0,R0]
packssdw  mm1,mm4     // mm4 = [G2,R2,B1,G1]
packssdw  mm2,mm5     // mm2 = [B3,G3,R3,B2]
pminsw    mm0, mm7    // mm0 =
[min(255,R1),min(255,B0),min(255,G0),min(255,R0)]
pminsw    mm1, mm7    // mm1 =
[min(255,G2),min(255,R2),min(255,B1),min(255,G1)]
pminsw    mm2, mm7    // mm2 =
[min(255,B3),min(255,G3),min(255,R3),min(255,B3)]
packuswb  mm0, mm1    // mm0 = [G2,R2,B1,G1,R1,B0,G0,R0]
packuswb  mm2, mm2    // mm2 = [*,*,*,*,B3,G3,R3,B2]
movq   [edi-12], mm0
movd   [edi-4], mm2

mov       _rgb, edi
pop       edi
add       esi, 16
add       eax, 16
add       edi, 16
dec       ecx
jnz       NextFourVectex
```

```
  // Data set up for the rest of vertices.
  DoRemainder:
  mov    ecx, reNum
  cmp    ecx, 0
  je   fininsh

  // Data set up for the rest of vertices.
  push edx
  mov    edx, 0FFFFFFh
  movd mm4, edx
  pop    edx

NextVectex:
  // one vertex for each iteration
  movss     xmm0, [esi]    // xmm0 = [*,*,*,nx]
  movss     xmm1, [eax]    // xmm1 = [*,*,*,ny]
  movss     xmm2, [edi]    // xmm2 = [*,*,*,nz]
  movss     xmm3, xmm0     // xmm3 = [*,*,*,nx]
  movss     xmm4, xmm1     // xmm4 = [*,*,*,ny]
  movss     xmm5, xmm2     // xmm5 = [*,*,*,nz]

  mulss     xmm0, [edx]    // xmm0 = [*,*,*,nx*lx]
  mulss     xmm1, [edx+16] // xmm1 = [*,*,*,ny*ly]
  mulss     xmm2, [edx+32] // xmm2 = [*,*,*,nz*lz]
  addss     xmm0, xmm1     // xmm0 = [*,*,*,nx*lx+ny*ly]
  addss     xmm0, xmm2     // xmm0 = [*,*,*,costheta]

  movss     xmm6, xmm0     // xmm6 = [costheta]

  addss     xmm0, xmm0     // xmm0 = [*,*,*,dt], dt = 2.0*costheta
  mulss     xmm3, xmm0     // xmm3 = [*,*,*,dt*nx]
  mulss     xmm4, xmm0     // xmm4 = [*,*,*,dt*ny]
  mulss     xmm5, xmm0     // xmm5 = [*,*,*,dt*nz]
  subss     xmm3, [edx]    // xmm3 = [*,*,*,dt*nx-lx]

  subss     xmm4, [edx+16] // xmm4 = [*,*,*,dt*ny-ly]

  subss     xmm5, [edx+32] // xmm5 = [*,*,*,dt*nz-lz]

  mulss     xmm3, [edx+48] // xmm3 = [*,*,*,Rx*vx]
  xorps     xmm7, xmm7     // clear xmm7
  mulss     xmm4, [edx+64] // xmm4 = [*,*,*,Ry*vy]
  mulss     xmm5, [edx+80] // xmm5 = [*,*,*,Rz*vz]
```

```
   subss     xmm7, xmm6     // xmm7 = [*,*,*,-costheta]
   addss     xmm3, xmm4     // xmm3 = [*,*,*,Rx*Vx+Ry*Vy]
   addss     xmm5, xmm3     // xmm5 = [*,*,*,cosalpha]
   maxss     xmm7, xmm6     // xmm7 = [*,*,*,Costheta], Costheta = |costheta|

   movss     xmm0,[edx+144] // xmm0 = [*,*,*,diffuseR]
   movss     xmm1,[edx+160] // xmm1 = [*,*,*,diffuseG]
   movss     xmm2,[edx+176] // xmm2 = [*,*,*,diffuseB]
   movss     xmm3,[edx+96]  // xmm3 = [*,*,*,ambientR]
   movss     xmm4,[edx+112] // xmm4 = [*,*,*,ambientG]
   movss     xmm6,[edx+128] // xmm6 = [*,*,*,ambientB]
   mulps     xmm0, xmm7     // xmm0 = [*,*,*,diffuseR*Costheta]
   mulps     xmm1, xmm7     // xmm1 = [*,*,*,diffuseG*Costheta]
   mulps     xmm2, xmm7     // xmm2 = [*,*,*,diffuseB*Costheta]

   addps     xmm0,xmm3   // xmm0 = [*,*,*,diffuseR*Costheta3+ambientR]
   addps     xmm1,xmm4   // xmm1 = [*,*,*,diffuseG*Costheta3+ambientG]
   addps     xmm2,xmm6   // xmm2 = [*,*,*,diffuseB*Costheta3+ambientB]
   xorps     xmm6, xmm6     // xmm6 = [0.0, 0.0, 0.0, 0.0]
   push   edi
   comiss    xmm5, xmm6     // check if cosAlpha < 0.0 ? (cosAlpha < 0.0 ==>
CF = 1)
   jc     Continue2

   movss     xmm7,[edx+192] // xmm7 = [*,*,*,specular]
   mov       edi, phongconst

loop21:
   mulss  xmm7, xmm5        // xmm7 = [specular*cosalpha3,..,..,..]
   dec    edi
   jnz    loop21

   addss     xmm0, xmm7     // xmm0 = [*,*,*,fr]
   addss     xmm1, xmm7     // xmm1 = [*,*,*,fg]
   addss     xmm2, xmm7     // xmm2 = [*,*,*,fb]
Continue2:
   mulss     xmm0, four_255 // xmm0 = [*,*,*,fR]
   mulss     xmm1, four_255 // xmm1 = [*,*,*,fG]
   mulss     xmm2, four_255 // xmm2 = [*,*,*,fB]

   mov       edi, _rgb
   cvtps2pi  mm0, xmm0   // mm0 = [*,R]
   cvtps2pi  mm1, xmm1   // mm1 = [*,G]
```

```
   cvtps2pi  mm2, xmm2   // mm2 = [*,B]
   punpckldq mm0, mm2    // mm0 = [B,R]
   packssdw  mm0, mm0    // mm0 = [*,*,B,R]
   punpcklwd mm0, mm1  // mm0 = [*,B,G,R]
   pminsw    mm0, mm7  // mm0 = [*,min(255,B),min(255,G),min(255,R)]
   packuswb  mm0, mm0    // mm0 = [*,*,*,*,*,B,G,R]

   // write three bytes R,G,B in mm0 to memory [edi]
   maskmovq  mm0, mm4
   add       edi, 3
   mov       _rgb, edi
   pop       edi
   add       esi, 4
   add       eax, 4
   add       edi, 4
   dec       ecx
   jnz       NextVectex

finish:

   emms

   }

}/*intensity_SOA_XMM() */
```